

# Machine learning

Uitgebreid onderzoek naar verschillende machine learning technieken heeft ons in staat gesteld om een geschikte oplossing te bedenken. Deze gaan niet alleen nauwkeurige voorspellingen doen, maar ook de juiste prioritering van incidenten mogelijk maken. Het machine learning aspect van het project is om de binnenkomende meldingen te classificeren volgens 'Urgentie'. Dit in een breder kader waar we Cipal Schaubroeck moeten helpen met het schaalbaar maken van hun ANPR-camera netwerk.

## Research en analyse

In dit deel van ons onderzoek duiken we dieper in de data. We zullen de data schoonmaken, analyseren en verschillende machine learning modellen onderzoeken. Door de data grondig te begrijpen, kunnen we waardevolle inzichten verkrijgen die ons helpen om de beste aanpak voor ons probleem te kiezen.

## Data

Om een betrouwbaar machine learning model te kunnen bouwen, moeten we eerst de data die we van Cipal hebben ontvangen grondig begrijpen en voorbereiden. Deze data bevat een schat aan informatie over verschillende incidenten, maar is nog niet in een vorm die direct geschikt is voor ons model.

### Data schoonmaken:

- **Overbodige kolommen verwijderen:** We doorlopen de data kolom voor kolom en verwijderen alle informatie die niet relevant is voor het voorspellen van incidenten.
- **Fouten corrigeren:** We controleren de data op fouten zoals typfouten of inconsistenties en corrigeren deze.
- **Data formatteren:** De data wordt gestandaardiseerd zodat alle waarden in hetzelfde formaat staan. Dit maakt het makkelijker voor het model om de data te verwerken.

Data structureren:

- **Kolommen aanpassen:** De namen van de kolommen maken we duidelijker, zodat we precies weten welke informatie elke kolom bevat.
- **Kolommen splitsen:** Als een kolom meerdere stukjes informatie bevat, splitsen we deze op in meerdere kolommen. Zo kunnen we bijvoorbeeld een kolom met datum en tijd opsplitsen in twee afzonderlijke kolommen.

Feature engineering:

- **Nieuwe kenmerken creëren:** Op basis van de bestaande data maken we nieuwe kenmerken aan die voor ons model relevant kunnen zijn. Bijvoorbeeld, uit een beschrijving van een incident kunnen we nieuwe kenmerken halen zoals de ernst van het incident of de categorie waartoe het behoort.

Om een beter beeld te krijgen van de data, gaan we verschillende visualisaties maken. Zo kunnen we patronen ontdekken en zien welke kenmerken belangrijk zijn voor onze analyse.

Dit is een voorbeeld van de data voordat we er iets aan hebben gedaan. De kolomnamen zijn onduidelijk en de data is niet goed gestructureerd.

event_datetime	event_host	host_dns	event_trigger	event_id	alert_subject	alert_message
26/07/2024	OUDSBERGENcam30 5602320 VDSL site 3OUNLIM419A	OUDSBERGENcam30.smvi.ciport.be	OUDSBERGENcam30 5602320 VDSL site 3OUNLIM419A ...	9027694	Problem - OUDSBERGENcam30 5602320 VDSL site 3O...	Problem started at 20:50:59 on 2024.07.26 Pro...

event_nseverity	event_value	topdesk_issue_key	topdesk_issue_link	trigger_id	recoverytime	event_recovery_time
4	1	{EVENT.TAGS.__zbx_tpd_issuekey}	{EVENT.TAGS.__zbx_tpd_issuelink}	33080	NaN	20:59:59

event_recovery_date	event_recovery_name	event_duration	camera_sn
26/07/2024	OUDSBERGENcam30 5602320 VDSL site 3OUNLIM419A ...	9m 0s	NaN

Hier zie je de data nadat we de preprocessing stappen hebben uitgevoerd. De kolomnamen zijn duidelijker, de data is gestructureerd en er zijn nieuwe kenmerken toegevoegd.

event_datetime	event_starting_time	event_solved_time	event_recovery_date	event_id	trigger_id
26/07/2024	20:50:59	20:59:59	26/07/2024	9027694	33080

host_dns	ip	issue	event_nseverity
OUDSBERGENcam30.smvi.ciport.be	10.134.95.194	is unavailable by ICMP	4

Dit kunnen we nog meer doen met onze data:

- **Ontbrekende waarden aanpakken:** We bepalen verder hoe we omgaan met ontbrekende waarden in de data.
- **Outliers verwijderen:** We identificeren en verwijderen extreme waarden die de resultaten van ons model kunnen beïnvloeden. (Geeks for geeks, 2024)
- **Data balanceren:** Als er een onevenwichtigheid is in de data, passen we technieken toe om de dataset te balanceren.
- **One-hot encoding:** Het omzetten van categorische gegevens naar numerieke gegevens met One-hot encoding (Geeks for geeks, 2024)

Hierover meer bij de sectie over bibliotheken.

## Inzichten

Door de data grondig te analyseren en voor te bereiden, leren we de data beter kennen. (Novogroder, 2024) Dit helpt ons om de juiste machine learning technieken te kiezen en om de resultaten van ons model beter te interpreteren.

1. Event\_datetime: De datum wanneer een foutmelding wordt gemeld. Deze komen niet altijd overeen met de datum waarop de melding wordt opgelost.
2. Event\_starting\_time: De tijd dat een foutmelding wordt vastgesteld.

3. Event\_solved\_time: De tijd dat de foutmelding wordt opgelost. Het komt soms voor dat het probleem niet binnen een uur is opgelost. Er zijn zelfs voorvallen waarop het meer dan 24 uur duurt voor een melding opgelost wordt.
4. Event\_recovery\_date: De datum wanneer een foutmelding is opgelost.
5. Event\_id: De id of nummer van een event.
6. Trigger\_id: De id of nummer van een trigger.
7. Host\_dns: De naam van de dns. Dit zou later nog onderverdeeld kunnen worden in locatie, camera nummer en domeinnaam.
8. Ip: Dit is het IP adres van de camera die de melding doorgaf.
9. Issue: Het probleem van de melding, deze is uit de event trigger gehaald.
10. Event\_nseverity: Dit is het nummer dat de ernst van de melding aangeeft. We hebben 4 soorten ernstigheid niveaus: 1 = informatief, 2 = waarschuwing, 3 = gemiddeld en 4 = hoog.

We gaan nog verder moeten onderzoeken welke datapunten, zowel individueel als in combinatie, de beste voorspellers zijn voor een optimaal model.

## Bibliotheken

Voor het realiseren van het machine learning gedeelte van ons project hebben we het gebruik van volgende bibliotheken overwogen: PyCaret, Keras, TensorFlow en PyTorch.

We gaan deze 4 vergelijken op basis van volgende criteria:

- **Gebruiksvriendelijkheid:** Aangezien we maar 3 weken hebben voor heel onze oplossing te realiseren, zitten we met een strakke deadline. Daarom zouden we graag werken met tools die gemakkelijk te begrijpen zijn door teamleden met een andere achtergrond waardoor zij kunnen helpen indien mogelijk.
- **Tijd tot Implementatie:** Zoals we besproken bij het vorige criteria, werken we met een strakke deadline. Daarom is het belangrijk dat we werken met tools die snel een efficiënt te implementeren zijn.
- **Documentatie:** Aangezien we nog niet ervaren zijn met deze tools willen we bibliotheken gebruiken die ondersteunt zijn met goede en uitgebreide documentatie.

- **Community-ondersteuning:** Het is voor ons belangrijk dat wanneer we vast komen te zitten, we kunnen rekenen op ondersteuning van de online community. Dit kan ons helpen bij snelheid van ontwikkeling.
- **Prestatie:** Onze oplossing zal honderden camera's en duizenden meldingen moeten verwerken per dag. Dat wilt zeggen dat de prestatie van onze gekozen bibliotheken en tools uitstekend moet zijn.
- **Ondersteuning voor use case:** Er zijn veel verschillende benodigdheden om onze oplossing te realiseren. Daarom moeten we zeker zijn dat onze gekozen bibliotheek deze benodigdheden tegemoet kan komen.

### *Mogelijke opties*

Eerst hebben we **TensorFlow**. TensorFlow is een end-to-end platform voor machine learning. TensorFlow is een open-source project dat eerst ontwikkelt is door het Google brain-team en nu onderhouden wordt door een grote community. TensorFlow is ontwikkelt om het trainen van neurale netwerken gemakkelijker te maken. (*TensorFlow*, 2024) TensorFlow heeft ook de mogelijkheid om een volledige ML pipeline uit te bouwen. Deze functionaliteiten zijn vervat in het TFX (TensorFlow Extended). TFX is een stuk van de TensorFlow bibliotheek die ontwikkelt is om te helpen bij de overgang van research naar deployment. (*TFX | ML Production Pipelines*, 2024)

Een andere mogelijke bibliotheek die we kunnen gebruiken is **PyCaret**. PyCaret is een low-code machine learning bibliotheek voor het automatiseren van machine learning workflows. PyCaret bouwt verder op andere bibliotheken zoals scikit-learn XGBoost, LightGBM, etc. PyCaret wilt machine learning bruikbaar maken voor de gemiddelde data analist. Daarom is elke deel van de bibliotheek ontworpen met gebruiksvriendelijkheid en gemak in het achterhoofd. PyCaret biedt bij installatie ook een brede waaier aan modellen aan om te gebruiken. Naast build-in modellen biedt PyCaret de mogelijkheden voor data preprocessing, model training, model analyse en model deployment. (*PyCaret 3.0 | Docs*, 2024)

Keras is een 3<sup>de</sup> mogelijkheid die we kunnen gebruiken. Keras was oorspronkelijk ontwikkeld als een zelfstandige bibliotheek en is tegenwoordig naadloos geïntegreerd in

TensorFlow. Zelfs met JAX en PyTorch is Keras makkelijk in gebruik. Deze tool heeft een lange instapdrempel maar heeft een steile leercurve. Omdat Keras zo nauw samenwerkt met TensorFlow is er een fantastische community met discussie blogs waar je al je vragen kan stellen aan elkaar. (Keras, 2024)

Een laatste optie is **PyTorch**. PyTorch is een open-source bibliotheek voor machine learning en deep learning, ontwikkeld door Meta AI (voorheen Facebook AI). PyTorch is een flexibele en krachtige tool die veel wordt gebruikt door onderzoekers en ontwikkelaars. Ze gebruiken dit voor het bouwen van neurale netwerken en andere geavanceerde ML-modellen. Net zoals TensorFlow, heeft PyTorch een grote en actieve community die bijdraagt aan de groei en ondersteuning van de bibliotheek.

Een van de onderscheidende kenmerken van PyTorch is het gebruik van dynamische computationele grafieken. Dit betekent dat het model tijdens runtime wordt gegenereerd, in tegenstelling tot vooraf gedefinieerde grafieken. Hierdoor is PyTorch intuïtiever voor debugging en het ontwikkelen van complexe modellen. (PyTorch, 2024)

Criteria	Gewicht	TensorFlow	PyCaret	Keras	PyTorch
Gebruiksvriendelijk	4	2	4	1	3
Implementatie Tijd	6	2	4	2	3
Documentatie	3	3	3	2	4
Community	1	4	2	4	3
Prestatie	2	4	2	3	3
Ondersteuning	5	3	4	2	2
Totaal	21	56	75	42	61

## Onderzoek PyCaret

Na het onderzoeken van de verschillende mogelijke bibliotheken die we zouden kunnen gebruiken hebben we gekozen voor PyCaret. Dit met de verschillende criteria in ons achterhoofd. PyCaret biedt ons de mogelijkheid om alles te doen in 1 tool en is gemaakt om gebruiksvriendelijk te zijn voor iedereen. Dit in combinatie met het low-code karakter verwachten we dat we op een versnelde manier onze oplossing kunnen creëren binnen onze deadline.

We hebben heel de opgave bekeken en op basis hiervan hebben we componenten van PyCaret gekozen om onderzoek naar te doen. Deze componenten, methoden en features worden in de volgende secties besproken. Zo hebben we het over verschillende features die PyCaret ondersteunt om data verder voor te bereiden voor training. We willen natuurlijk ook modellen kiezen, deze trainen en hieruit een besluit nemen. Als laatste nemen we een kijkje naar hoe we het best getrainde model kunnen evalueren, debuggen en uiteindelijk deployen voor gebruik.

## Data Preprocessing

Zoals eerder al besproken hebben we de data al geformatteerd maar er zijn nog een aantal zaken die we nog moeten aanpakken. Zo zijn er 'missing values' in onze dataset. Missing values zijn waarden die missen in onze dataset. Naast missing values willen we ook uitschieters bekijken en eventueel verwijderen en de data balanceren. Uitschieters (zoals de naam al verklapt) zijn waarde die extreem hoog of extreem laag zijn. (Ellis, 2021) Data balanceren is dan weer een activiteit die ervoor zorgt dat er van alle trainingsdata even veel is. Dat we een model dus niet onnodig veel geven van label 1 en minder van label 2. (*Datasets: Imbalanced Datasets*, 2024) Als laatste willen we onze data ook 'one-hot' encoden. We hebben namelijk 4 soorten meldingen: information, warning, average en high. We willen deze daarom omvormen naar numerieke waarden om deze soorten duidelijk te maken voor het model. One-hot encoding gaat als het waren 4 kolommen toevoegen (een voor elk soort). Wanneer dan een melding bijvoorbeeld 'high' is, zal de

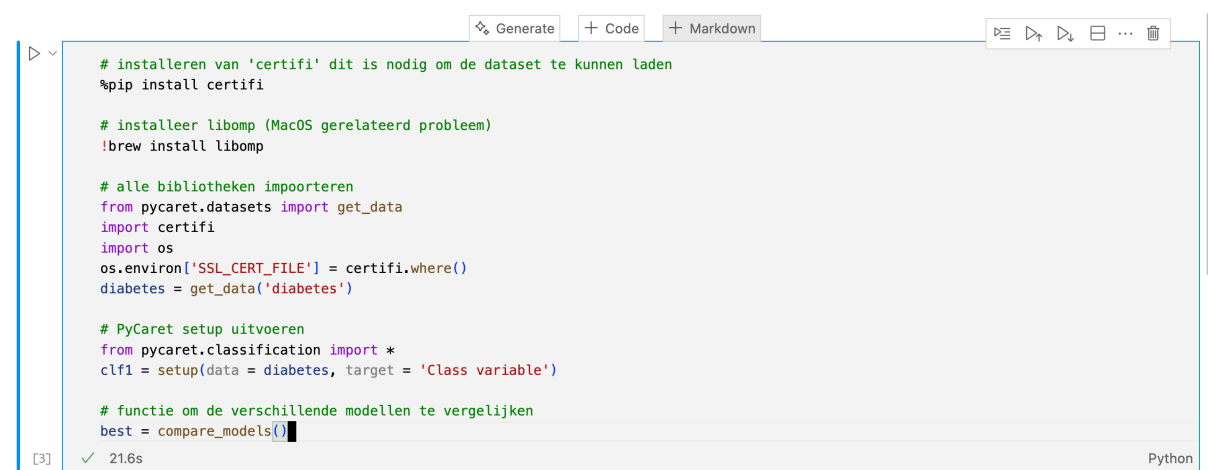
kolom voor 'high' 1 bevatten en alle andere 0. (*One-Hot Encoding Explained*, 2024) Deze techniek gaan we ook moeten toepassen op de issues kolom.

Hieronder verkennen we de mogelijkheid van deze functionaliteiten in code. Bij elke lijn code is in commentaar bijgevoegd hoe we dit zouden doen. Hier gebruikte we een PyCaret dataset om dit te testen.

## Model Training

We hebben tijdens de Concept fase onze data al wat opgeschoond en verstaanbaar gemaakt. Maar we hebben nog niet gekozen welk model we best kunnen gebruiken. Daar kan PyCaret ons ook mee helpen. Het heeft namelijk een functie die we kunnen gebruiken om alle verschillende modellen te rangschikken op basis van de prestaties. (*Train | Docs*, 2024) Het gaat accuratie, AUC, Recall, precisie, F1, Kappa MCC en TT vergelijken met elkaar en de verschillende modellen daarop beoordelen. (*Metrics for Classification: Accuracy, Precision, Recall, F1-Score, ROC-AUC | Modlee*, 2024)

Hieronder zie je een stuk code dat we tijdens research hebben geschreven over hoe we deze functie kunnen gebruiken. Deze code bepaalt welke modellen we kunnen gebruiken. Deze code gebruikt dezelfde dataset als we gebruikten bij Data Preprocessing.



```
# installeren van 'certifi' dit is nodig om de dataset te kunnen laden
!pip install certifi

# installeer libomp (MacOS gerelateerd probleem)
!brew install libomp

# alle bibliotheken importeren
from pycaret.datasets import get_data
import certifi
import os
os.environ['SSL_CERT_FILE'] = certifi.where()
diabetes = get_data('diabetes')

# PyCaret setup uitvoeren
from pycaret.classification import *
clf1 = setup(data = diabetes, target = 'Class variable')

# functie om de verschillende modellen te vergelijken
best = compare_models()
```

[3] ✓ 21.6s Python



	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
ridge	Ridge Classifier	0.7654	0.8255	0.5547	0.7241	0.6187	0.4545	0.4692	0.0040
lda	Linear Discriminant Analysis	0.7654	0.8253	0.5599	0.7199	0.6201	0.4554	0.4695	0.0040
lr	Logistic Regression	0.7617	0.8240	0.5599	0.7108	0.6169	0.4486	0.4615	0.3220
gbc	Gradient Boosting Classifier	0.7561	0.8211	0.5827	0.6736	0.6221	0.4441	0.4486	0.0200
et	Extra Trees Classifier	0.7505	0.8073	0.5602	0.6785	0.6042	0.4263	0.4366	0.0190
rf	Random Forest Classifier	0.7487	0.8198	0.5345	0.6851	0.5960	0.4180	0.4278	0.0250
nb	Naive Bayes	0.7468	0.8111	0.5974	0.6626	0.6196	0.4316	0.4397	0.0040
ada	Ada Boost Classifier	0.7393	0.7886	0.5553	0.6489	0.5939	0.4047	0.4102	0.0130
qda	Quadratic Discriminant Analysis	0.7374	0.8096	0.5439	0.6604	0.5871	0.3983	0.4090	0.0050
lightgbm	Light Gradient Boosting Machine	0.7300	0.7886	0.5664	0.6398	0.5948	0.3941	0.4004	0.2530
knn	K Neighbors Classifier	0.7205	0.7558	0.5550	0.6130	0.5787	0.3714	0.3747	0.0090
dt	Decision Tree Classifier	0.7191	0.6941	0.6111	0.5934	0.5993	0.3839	0.3862	0.0040
dummy	Dummy Classifier	0.6518	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0040
svm	SVM - Linear Kernel	0.6163	0.5774	0.4126	0.3925	0.3612	0.1389	0.1573	0.0060

De output van deze ‘compare\_models’ methode is hieronder zichtbaar. Hier kunnen we zien dat het ‘Ridge Classifier’ model het beste scoort voor de gebruikte dataset.

```
# train het ridge Classifier model
ridge = create_model('ridge', fold=3)
```

[2] ✓ 0.1s Python

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
Fold							
0	0.7654	0.8198	0.6129	0.6786	0.6441	0.4697	0.4711
1	0.7430	0.7956	0.4677	0.6905	0.5577	0.3859	0.4004
2	0.7598	0.8317	0.5397	0.7083	0.6126	0.4431	0.4517
Mean	0.7561	0.8157	0.5401	0.6925	0.6048	0.4329	0.4411
Std	0.0095	0.0150	0.0593	0.0122	0.0357	0.0350	0.0298

## Model Evaluation

Na het kiezen van ons model en het trainen zouden we graag evalueren hoe het model presteert. Hiervoor heeft PyCaret weer ingebouwde functionaliteiten om dit gemakkelijker te maken voor ons. Eerst en vooral zouden we dan gebruik kunnen maken van een methode om het model te evalueren. ‘Evaluate\_model’ creëert een gemakkelijk te gebruiken user interface. Het laat zien, aan de hand van verschillende grafieken, hoe het model presteert. Zo kunnen wij afleiden hoe we het model verder kunnen verbeteren. (Analyse, 2024) Daarnaast kunnen we ook de output van een model gaan analyseren. Dit door de methode ‘interpret\_model’ te gebruiken. De voorspellingen van het model analyseren en weergeven in grafieken gebaseerd op SHAP (SHapley Additive exPlanations). (Analyse, 2024) Zo biedt het inzicht in hoe elk kenmerk van een model bijdraagt aan een specifieke voorspelling. (Aboze, 2024) Als laatste willen we ook graag zeker zijn dat het model geen inherente bias heeft tegenover een bepaalde groep. We willen dus nagaan of het model van zichzelf niet een bepaalde groep, melding categorie,

individu... voortrekt of nadelig behandelt. Dit kunnen we checken met een PyCaret methode genaamd ‘check\_fairness’. (Analyse, 2024) Hieronder kan u de output vinden van deze 3 methoden. Deze willen we gebruiken om een algemeen overzicht te krijgen van de prestatie van ons gekozen model.

Deze code is nogmaals een fragment van onderzoek naar PyCaret. Deze code bouwt verder op de code die hierboven getoond wordt.

```
# Model evalueren
evaluate_model(ridge)

# Model interpreteren (dit werkt alleen bij modellen die SHAP ondersteunen dus niet ridge classifier)
# interpret_model(lr)

# Model controleren op eerlijkheid
check_fairness(ridge, sensitive_features="Number of times pregnant")
```

[4] ✓ 1.0s Python

Plot Type:

Pipeline Plot

Hyperparameters

AUC

Confusion Matrix

Threshold

Precision Recall

Prediction Error

Class Report

Feature Selection

Learning Curve

Manifold Learning

Calibration Curve

Validation Curve

Dimensions

Feature Importance

Feature Importance...

Decision Boundary

Lift Chart

Gain Chart

Decision Tree

KS Statistic Plot

Raw data

SimpleImputer

SimpleImputer

CleanColumnNames

RidgeClassifier

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Ridge Classifier	0.7359	0.6717	0.4568	0.6852	0.5481	0.3720	0.3872

	Samples	Accuracy	Recall	Precision	F1	Kappa	MCC	Selection Rate
Number of times pregnant								
0	31.0	0.806452	0.545455	0.857143	0.666667	0.539604	0.566971	0.225806
1	45.0	0.866667	0.285714	0.666667	0.400000	0.338235	0.376897	0.066667
2	37.0	0.702703	0.181818	0.500000	0.266667	0.128480	0.154401	0.108108
3	28.0	0.821429	0.666667	0.750000	0.705882	0.578313	0.580381	0.285714
4	15.0	0.866667	0.666667	1.000000	0.800000	0.705882	0.738549	0.266667
5	16.0	0.625000	0.285714	0.666667	0.400000	0.186441	0.221917	0.187500
6	13.0	0.615385	0.400000	0.500000	0.444444	0.155844	0.158114	0.307692
7	14.0	0.571429	0.500000	0.666667	0.571429	0.160000	0.166667	0.428571
8	12.0	0.583333	0.285714	1.000000	0.444444	0.250000	0.377964	0.166667
9	8.0	0.625000	0.600000	0.666667	0.250000	0.258199	0.500000	0.500000
10	4.0	0.000000	0.000000	0.000000	0.000000	-0.600000	-1.000000	0.750000
11	3.0	0.666667	1.000000	0.666667	0.800000	0.000000	0.000000	1.000000
12	1.0	1.000000	0.000000	0.000000	0.000000	NaN	0.000000	0.000000
13	3.0	0.666667	1.000000	0.500000	0.666667	0.400000	0.500000	0.666667
14	1.0	1.000000	1.000000	1.000000	1.000000	NaN	0.000000	1.000000



## Model Deployment

Als laatste willen we ons model natuurlijk ook gebruiken in onze applicatie. Aangezien PyCaret een end-to-end bibliotheek is, hebben ze voor het deployen ook ingebouwde methoden. Als we het model willen gebruiken moeten we het de mogelijkheid geven om het te laten communiceren met de andere applicaties. Dit doen we door een API rond het model te creëren. Daarna moeten we de API openstellen naar de buiten wereld, dit kunnen we doen door de API te containeriseren en online te hosten op een Cloud provider. Hiervoor biedt PyCaret ‘create\_api’ en ‘create\_docker’ methoden aan. De methode om een API te creëren maakt automatisch een API POST endpoint aan. Dat wil zeggen dat vanaf we de API opstarten, we kunnen connecteren. Dit houdt in dat we relevante data er naartoe kunnen sturen en een antwoord zullen terugkrijgen. Wanneer we data terugkrijgen kunnen we de create\_docker methode gebruiken om automatisch een Dockerfile en requirements bestand aan te maken. (*Deploy | Docs, 2024*) Zo kunnen we op een geautomatiseerde manier Docker images aanmaken en deze naar de Dockerhub registry sturen. Zo kunnen we voor de gehele ML pipeline een Continuous Deployment opzetten.

Deze code is een fragment van onderzoek naar PyCaret. Deze code bouwt verder op de code die hierboven getoond wordt.



```
# API creëren rond het model
create_api(ridge, 'diabetes_api')

# API starten
!python diabetes_api.py

# Dockerfile en requirements bestand aanmaken
create_docker('diabetes_api')

# Docker image maken
!docker image build -f "Dockerfile" -t test_api:latest .
```

Python

## Oplossing

### Verloop

De data hebben we alvast voor een deel opgeschoond (verstaanbaar en klaar voor interpretatie). Het is echter nog niet volledig klaar. Er staan doorheen de data nog 'null' values en andere inconsistenties. Daarom gaan we bij het begin van onze 3 weken aan sprints starten met de data training klaar te maken. Dit doen we door de besproken functionaliteiten van PyCaret te gebruiken.

Wanneer dit gebeurt is kunnen we starten met het kiezen en trainen van ons model. Zo willen we de methoden van PyCaret gebruiken om onze training klare data te trainen op verschillende modellen. Daarna bekijken we de door PyCaret geven scoren van deze modellen. Zo weten we direct welk model de beste prestaties zal leveren. Na het beslissen welke criteria belangrijker zijn voor onze toepassing gaan we naar het trainen van ons gekozen model.

Na het trainen van het model kunnen we al bijna met zekerheid zeggen dat het nog niet perfect zal werken. Daarom zal er een periode ingerekend worden om de prestaties van het model te analyseren, te verfijnen en te her-trainen. Dit zullen we doen met de bovenaan besproken methoden van PyCaret.

Wanneer we dit doen, kan een van de (AI) teamleden van start gaan met het maken van de API. Zoals hierboven zullen we een API creëren om het model de mogelijkheid te geven tot communiceren met andere applicaties. We zullen hiervoor nogmaals PyCaret gebruiken. PyCaret maakt voor ons een API gebaseerd op FastAPI (op python gebaseerd

framework) (*FastAPI*, 2024). Wanneer dit alles werkt kunnen we met een simpele methode van PyCaret een Dockerfile genereren. Deze Dockerfile kan gebruikt worden om een Docker image te maken die dan naar de Dockerhub kan gestuurd worden. (“*Docker Push*,” 2024) Deze image kan dan door AWS automatisch van de Dockerhub gehaald worden om een nieuwe instantie van de API te hosten. (*Authenticating With Docker Hub for AWS Container Services* | *Amazon Web Services*, 2021)

## Extra's

De volgende stap in ons onderzoek is het verrijken van onze dataset met aanvullende informatie. Door sensordata zoals temperatuur, vochtigheid en beweging toe te voegen, kunnen we een meer gedetailleerd beeld krijgen van de omgeving en de invloed hiervan. Dit zal een grote impact hebben op de gebeurtenissen die we proberen te voorspellen.

We zullen een nieuwe dataset samenstellen die de sensorgegevens bevat. Deze dataset zal vervolgens worden gebruikt om een nieuw model te trainen, waarmee we de nauwkeurigheid van onze voorspellingen hopelijk verder kunnen verbeteren.

## Risicobeheersing

Zo kunnen we ervoor zorgen dat er zo min mogelijk misgaat.

- **Data kwaliteit:** We zorgen voor een goede data kwaliteit door regelmatig de data te controleren op fouten, inconsistenties en ontbrekende waarden.
- **Regelmatige evaluatie:** We gaan regelmatig evaluaties uitvoeren op de prestaties van ons model om afwijkingen vroegtijdig te detecteren.
- **Re-training:** We gaan het model regelmatig opnieuw trainen met nieuwe data om de nauwkeurigheid te behouden en aanpassingen aan de dataverdeling te verwerken.
- **Alternatieve bronnen:** Tijdens onze research hebben we alternatieve bronnen voor bibliotheken bekeken. Deze kunnen we toepassen in geval van problemen.

## Conclusie

Onze bevindingen benadrukken het belang van datakwaliteit en feature engineering om optimale modelprestaties te bereiken. Daarnaast zijn de flexibiliteit en efficiëntie van PyCaret een goede match voor onze toepassing. Deze tool heeft alle nodige methoden die we willen gebruiken.

In de volgende fase gaan we onze research omzetten naar een werkend model. Deze verfijnen we zodat de voorspelling optimaal zijn en we verder kunnen met de volgende dataset. De vernieuwde dataset zal data bevatten van onze IoT toepassing waardoor het modeltraining en verfijningsproces terug opnieuw moet gebeuren.

Door onze oplossing continu te monitoren en bij te werken, streven we ernaar om proactief potentiële problemen te identificeren en te beperken. Hierdoor zorgen we voor voortdurende betrouwbaarheid en efficiëntie van onze modellen.

Voor alle bronnen te raadplegen, stuur mij een mail via [r0929448@student.thomasmore.be](mailto:r0929448@student.thomasmore.be). Dan bezorg ik het volledige projectplan. Dit is namelijk een verkorte versie van het volledige plan.